Rigorous || Structured || Proof-Oriented

Refoma Core | Foundations of Mathematics | 2026

ALGORITHMICS ONE

This book introduces algorithmic thinking through fundamental concepts such as variables, control structures, and iterative processes. It emphasizes systematic logical problem-solving, supported by practical exercises and clearly explained solutions.

Abdeljalil Mostefaoui | Echahid Cheikh Larbi Tebessi university

Algorithms & Data Structures 01

Refoma Core — Algorithms

Course Summary & Solved Exercises

Subject: Algorithms& Data Structures I
Educational level: 1st Year University mathematics
Student: Abdeljalil Mostefaoui

Algorithm Note:
"Every algorithm transforms complexity into structured simplicity."

Designed to support students in mastering the fundamental concepts of Algorithms.

Abdeljalil Mostefaoui | Echahid Cheikh Larbi Tebessi university

First Edition

Copyright © 2026 by Abdeljalil Mostefaoui.
All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the author, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.
Typeset in LaTeX.

Dedication

All praise and gratitude are due to Allah, the Almighty, whose guidance, mercy, and blessings have made this work possible. Through His wisdom and support, I was granted the strength, patience, and determination required to pursue knowledge and to complete this academic journey.

I dedicate this work with profound respect, love, and gratitude to my parents, who have been the cornerstone of my education. To my beloved father, **Abdelhamid Mostefaoui**, whose unwavering dedication, sacrifices, and moral guidance have always inspired me to strive for excellence. His constant encouragement, wisdom, and belief in the value of education have shaped my character and motivated me to persevere through challenges.

To my dear mother, **Mebarka.W.Z**, whose endless love, patience, and prayers have been my greatest source of strength. Her quiet sacrifices, sincere support, and unconditional care have illuminated my path and provided me with the confidence and serenity necessary to pursue my studies with determination and humility. Words cannot fully express my gratitude for her role in every achievement I reach.

This work is a humble expression of gratitude to all those who believed in me, supported me, and contributeddirectly or indirectlyto my educational journey. May this achievement reflect a small part of their sacrifices, efforts, and noble intentions.

Abdeljalil Mostefaoui

Contents

1 Course Cheat Sheet & Roadmap **5**
1.1 Chapter 1: Basic Instructions & Loops 5
1.2 Chapter 2: Arrays 6
1.3 Chapter 3: Strings 6
1.4 Chapter 4: Structures 7

2 The Exercise **8**

3 The Exam Test **9**

4 The Solutions **10**
4.1 I. Basic Instructions (1-10) 10
4.2 II. Arrays (11-18) 20
4.3 III. Character Strings (19-24) 28
4.4 IV. Structures / Records (25-30) 34

5 The Exam Solutions **40**

1 Course Cheat Sheet & Roadmap

This section summarizes the key concepts for each chapter and maps them to the relevant revision exercises. Click on the exercise number to jump to it.

Quick Navigation Guide

Click on exercise numbers to jump directly to detailed solutions.
Use this sheet as your revision roadmap throughout the course.

1.1 Chapter 1: Basic Instructions & Loops

Core Concepts

Variables & Data Types:

- `int`: Integers (e.g., 5, -10, 0)
- `float`: Real numbers (e.g., 3.14, -2.5)
- `char`: Single characters (e.g., 'A', '9')
- `double`: High-precision floats

Loops Summary:

- `for`: When iterations are known
- `while`: Check condition first
- `do-while`: Execute at least once

Essential Syntax

Operation	Syntax
Input	`scanf("%d", &x);`
Output	`printf("Value: %d", x);`
If-Else	`if(x > 0) {...} else {...}`
For Loop	`for(i=0; i<n; i++) {...}`
While Loop	`while(x != 0) {...}`
Do-While	`do {...} while(x < 10);`

Key Algorithms:

- **Conditional Logic:** Comparisons, parity checks, sorting
- **Accumulation:** Sums, products, factorials
- **Number Manipulation:** Digit operations, reversals

Practice Exercises:

Category	Exercises	Skills Tested
Conditionals	1, 2, 3, 4	Decision making
Loops	5, 6, 7	Iteration control
Math Logic	8, 9, 10	Problem solving

1.2 Chapter 2: Arrays

Core Concepts

Array Basics:

- `int T[10]; // 10-element array`
- Index range: 0 to N-1
- Memory: Contiguous allocation
- Initialization possible

Traversal Pattern:

```
for(i = 0; i < N; i++) {
  // Access T[i]
}
```

Essential Algorithms

Sum/Average:

```
sum = 0;
for(i=0; i<N; i++)
sum += T[i];
avg = sum/N;
```

Min/Max:

```
min = max = T[0];
for(i=1; i<N; i++) {
  if(T[i] < min)
min=T[i];
  if(T[i] > max)
max=T[i];
  }
```

Search:

```
found = 0;
for(i=0; i<N; i++) {
  if(T[i] == x) {
    found = 1; break;
  }
}
```

Key Patterns:

- **Aggregation**: Sum, average, count
- **Searching**: Linear search, existence check
- **Manipulation**: Copy, reverse, merge
- **Analysis**: Order checking, statistics

Practice Exercises:

Category	Exercises	Skills Tested
Basics	11, 12, 14	Aggregation, Statistics
Logic	13, 15	Analysis, Searching
Manipulation	16, 17, 18	Array operations
Exam	1, 3	Complex problems

1.3 Chapter 3: Strings

Core Concepts

String Definition:

- Character array
- Ends with `\0`
- Must include space for null
- `char S[100];`

Input Methods:

- `scanf("%s", S)` - No spaces
- `fgets(S, size, stdin)` - Safe
- `gets(S)` - Unsafe (avoid)

ASCII Reference

Char	ASCII	Char	ASCII
'A'	65	'a'	97
'0'	48	' '	32
'\0'	0	'\n'	10

Case Conversion:

- To upper: `ch - 32`
- To lower: `ch + 32`
- Use `toupper()`, `tolower()`

Key Algorithms:

- **Manual Functions**: strlen, strcpy, strcmp
- **Text Analysis**: Vowel counting, palindrome check
- **Manipulation**: Case conversion, validation

Practice Exercises:

Category	Exercises	Skills Tested
Manual Functions	19, 22, 23	String operations
Text Analysis	20, 21	Pattern recognition
Exam Practice	2, 5	Applied problems

1.4 Chapter 4: Structures

Core Concepts

Structure Definition:

```
typedef struct {
  type1 field1;
  type2 field2;
  // ...
} StructName;
```

Access Operators:

- Dot operator: `s.field`
- Arrow operator: `ptr->field`
- For arrays: `array[i].field`

Common Patterns

Declaration:

```
Student s1;
```

Input:

```
scanf("%s", s1.name);
scanf("%f", &s1.avg);
```

Array:

```
Student class[30];
class[i].name
```

Sorting:

```
if(class[i].avg < class[j].avg)
  swap(class[i], class[j]);
```

Key Applications:

- **Data Grouping**: Student records, contacts
- **Database-like Operations**: Search, sort, filter
- **Real-world Modeling**: Phonebook, inventory

2 The Exercise

I. Basic Instructions ($1 \rightarrow 10$)

1. Write an algorithm and a C program that reads two integers and prints the greater one. **[Solution]**
2. Read an integer and display whether it is even or odd. **[Solution]**
3. Read a real number x and display: x^2 if $x \geq 0$, otherwise $|x|$. **[Solution]**
4. Read three integers and display them in ascending order. **[Solution]**
5. Compute the sum of the first N integers (N given). **[Solution]**
6. Read an integer and display its multiplication table. **[Solution]**
7. Read an integer n and compute $n!$ (factorial). **[Solution]**
8. Read an integer and check whether it is a prime number. **[Solution]**
9. Read an integer and display the number of digits. **[Solution]**
10. Read an integer and display its reverse (example: $123 \rightarrow 321$). **[Solution]**

II. Arrays ($11 \rightarrow 18$)

11. Read an array of 10 integers and display the sum and the average. **[Solution]**
12. Read an array and determine the maximum and minimum values. **[Solution]**
13. Check whether an array is sorted in ascending order. **[Solution]**
14. Count the number of positive, negative, and zero values in an array. **[Solution]**
15. Search for a value in an array (sequential search). **[Solution]**
16. Copy an array A into another array B. **[Solution]**
17. Reverse the elements of an array. **[Solution]**
18. Merge two arrays into a third one. **[Solution]**

III. Character Strings ($19 \rightarrow 24$)

19. Read a string and display its length (without using `strlen`). **[Solution]**
20. Count the number of vowels in a string. **[Solution]**
21. Check whether a string is a palindrome. **[Solution]**
22. Copy a string into another one (without using `strcpy`). **[Solution]**
23. Compare two strings (without using `strcmp`). **[Solution]**
24. Convert a string to uppercase. **[Solution]**

IV. Structures / Records ($25 \rightarrow 30$)

25. Define a structure `Student` (name, first name, average) and display its information. **[Solution]**
26. Read the data of N students and display the list. **[Solution]**
27. Search for a student by name. **[Solution]**
28. Find the student with the highest average. **[Solution]**
29. Sort an array of student structures by average. **[Solution]**
30. Manage a simple directory using a structure `Contact` (name, phone number) with add and display operations. **[Solution]**

3 The Exam Test

EXERCISE 1 - Array Order & Analysis (4 pts) [Solution]
Write an algorithm and a C program that:

- Reads an integer array of size N.
- Determines whether the array is: strictly increasing, strictly decreasing, or not ordered.
- Displays the result.

EXERCISE 2 - Longest Word in a Sentence (4 pts) [Solution]
Write a C program that:

- Reads a sentence (one line).
- Finds and displays: the longest word and its length.
- Words are separated by spaces.

EXERCISE 3 - Frequency Table (3 pts) [Solution]
Write a program that:

- Reads an array of integers (values between 0 and 100).
- Builds a frequency table.
- Displays all values that appear more than once.

EXERCISE 4 - Student Records (5 pts) [Solution]
Define a structure `Student` containing: ID, Name, Average grade.
Write a program that:

- Reads data for N students.
- Displays students whose average ≥ 10.
- Calculates and displays the class average.
- Finds the best student.

EXERCISE 5 - String Validation (4 pts) [Solution]
Write a program that checks whether a given string is a valid identifier in C:

- Rules: Starts with a letter or _. Contains only letters, digits, or _. No spaces allowed.
- Display Valid or Invalid.

4 The Solutions

4.1 I. Basic Instructions (1-10)

Solution

1. Max of Two Numbers

[Back]

Algorithm

Steps:

1. Start
2. Read two numbers: `a` and `b`
3. If `a > b` then:
   - `max = a`
4. Else:
   - `max = b`
5. Print `max`
6. End

Explanation: This is a basic conditional statement. We compare the two numbers using the `if-else` structure.

C Code:

```c
#include <stdio.h>
int main() {
  int a, b, max;
  printf("Enter two numbers: ");
  scanf("%d %d", &a, &b);

  if(a > b)
  max = a;
  else
  max = b;

  printf("Maximum = %d\n", max);
  return 0;
}

```

Solution

2. Even or Odd [Back]

Algorithm

Steps:

1. Start
2. Read integer `n`
3. If `n % 2 == 0` then:
   - Print "Even"
4. Else:
   - Print "Odd"
5. End

Explanation: Uses the modulus operator (%) to check divisibility by 2. If remainder is 0, the number is even.

C Code:

```c
#include <stdio.h>
int main() {
  int n;
  printf("Enter an integer: ");
  scanf("%d", &n);

  if(n % 2 == 0)
  printf("%d is Even\n", n);
  else
  printf("%d is Odd\n", n);

  return 0;
}

```

Solution

3. Real Number Operation

[Back]

Algorithm

Steps:

1. Start
2. Read real number `x`
3. If `x >= 0` then:
   - `result = x * x`
4. Else:
   - `result = -x` (absolute value)
5. Print `result`
6. End

Explanation: For non-negative numbers, we square it. For negative numbers, we take the absolute value.

C Code:

```c
#include <stdio.h>
#include <math.h>  // for fabs if needed
int main() {
  float x, result;
  printf("Enter a real number: ");
  scanf("%f", &x);

  if(x >= 0)
  result = x * x;
  else
  result = -x;  // or result = fabs(x);

  printf("Result = %.2f\n", result);
  return 0;
}

```

Solution

4. Sort 3 Integers (Ascending)

[Back]

Algorithm

Steps:

1. Start
2. Read three integers: `a, b, c`
3. Compare and swap if necessary:
   - If `a > b`, swap them
   - If `a > c`, swap them
   - If `b > c`, swap them
4. Print `a, b, c` (now in ascending order)
5. End

Explanation: Uses bubble sort logic with three comparisons. After sorting, `a` is smallest, `c` is largest.

C Code:

```c
#include <stdio.h>
int main() {
  int a, b, c, temp;
  printf("Enter three integers: ");
  scanf("%d %d %d", &a, &b, &c);

  // Sort in ascending order
  if(a > b) { temp = a; a = b; b = temp; }
  if(a > c) { temp = a; a = c; c = temp; }
  if(b > c) { temp = b; b = c; c = temp; }

  printf("Sorted: %d %d %d\n", a, b, c);
  return 0;
}

```

Solution

5. Sum of First N Integers

[Back]

Algorithm

Steps:

1. Start
2. Read integer `N`
3. Initialize `sum = 0`
4. For `i = 1` to `N`:
   - `sum = sum + i`
5. Print `sum`
6. End

Explanation: Uses a `for` loop to accumulate the sum. Could also use formula: `sum = N*(N+1)/2`.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, sum = 0;
  printf("Enter N: ");
  scanf("%d", &N);

  for(i = 1; i <= N; i++) {
    sum += i;
  }

  printf("Sum of first %d integers = %d\n", N, sum);
  return 0;
}

```

Solution

6. Multiplication Table

[Back]

Algorithm

Steps:

1. Start
2. Read integer `n`
3. For `i = 1` to 10:
   - `result = n * i`
   - Print "`n` * `i` = `result`"
4. End

Explanation: Simple loop that multiplies the number by each integer from 1 to 10.

C Code:

```c
#include <stdio.h>
int main() {
  int n, i;
  printf("Enter a number: ");
  scanf("%d", &n);

  printf("Multiplication table of %d:\n", n);
  for(i = 1; i <= 10; i++) {
    printf("%d * %d = %d\n", n, i, n * i);
  }

  return 0;
}

```

Solution

7. Factorial [Back]

Algorithm

Steps:

1. Start
2. Read integer `n`
3. Initialize `fact = 1`
4. For `i = 1` to `n`:
   - `fact = fact * i`
5. Print `fact`
6. End

Explanation: Factorial of n (n!) = 1 Œ 2 Œ 3 Œ ... Œ n. For n=0, factorial is 1.

C Code:

```c
#include <stdio.h>
int main() {
  int n, i;
  unsigned long long fact = 1;  // Use long long for large numbers

  printf("Enter a positive integer: ");
  scanf("%d", &n);

  if(n < 0) {
    printf("Factorial doesn't exist for negative numbers.\n");
  } else {
    for(i = 1; i <= n; i++) {
      fact *= i;
    }
    printf("Factorial of %d = %llu\n", n, fact);
  }

  return 0;
}
```

Solution

8. Prime Number Check

[Back]

Algorithm

Steps:

1. Start
2. Read integer `n`
3. If `n < 2`:
   - Not prime
4. Else:
   - For `i = 2` to `sqrt(n)`:
   - If `n % i == 0`:
     - Not prime, break
5. If no divisors found, number is prime
6. End

Explanation: A prime number is only divisible by 1 and itself. We check divisors up to n for efficiency.

C Code:

```c
#include <stdio.h>
#include <math.h>
int main() {
  int n, i, isPrime = 1;

  printf("Enter a positive integer: ");
  scanf("%d", &n);

  if(n <= 1) {
    isPrime = 0;
  } else {
    for(i = 2; i <= sqrt(n); i++) {
      if(n % i == 0) {
        isPrime = 0;
        break;
      }
    }
  }

  if(isPrime)
  printf("%d is a prime number.\n", n);
  else
  printf("%d is NOT a prime number.\n", n);

  return 0;
}

```

Solution

9. Count Digits in Integer

[Back]

Algorithm

Steps:

1. Start
2. Read integer `n`
3. Initialize `count = 0`
4. If `n == 0`:
   - `count = 1`
5. Else:
   - While `n != 0`:
   - `n = n / 10`
   - `count++`
6. Print `count`
7. End

Explanation: Continuously divide by 10 until number becomes 0. Each division removes one digit.

C Code:

```c
#include <stdio.h>
int main() {
  int n, count = 0;

  printf("Enter an integer: ");
  scanf("%d", &n);

  // Handle 0 separately
  if(n == 0) {
    count = 1;
  } else {
    // Handle negative numbers
    if(n < 0) n = -n;

    while(n != 0) {
      n /= 10;    // Remove last digit
      count++;    // Increment counter
    }
  }

  printf("Number of digits = %d\n", count);
  return 0;
}
```

Solution

10. Reverse an Integer [Back]

Algorithm

Steps:

1. Start
2. Read integer `n`
3. Initialize `reverse = 0`
4. While `n != 0`:
   - `lastDigit = n % 10`
   - `reverse = reverse * 10 + lastDigit`
   - `n = n / 10`
5. Print `reverse`
6. End

Explanation: Extract last digit using %, add it to reversed number, then remove it using division.

C Code:

```c
#include <stdio.h>
int main() {
  int n, reverse = 0, remainder;

  printf("Enter an integer: ");
  scanf("%d", &n);

  int original = n;  // Keep original for display

  // Handle negative numbers
  if(n < 0) n = -n;

  while(n != 0) {
    remainder = n % 10;           // Get last digit
    reverse = reverse * 10 + remainder;  // Add to reversed number
    n /= 10;                      // Remove last digit
  }

  // Restore sign for negative numbers
  if(original < 0) reverse = -reverse;

  printf("Reverse of %d = %d\n", original, reverse);
  return 0;
}
```

4.2 II. Arrays (11-18)

Solution

11. Array Sum and Average

[Back]

Algorithm

Steps:

1. Start
2. Read size `N`
3. Initialize array `T[N]` and `sum = 0`
4. For `i = 0` to `N-1`:
   - Read `T[i]`
   - `sum = sum + T[i]`
5. `average = sum / N`
6. Print `sum` and `average`
7. End

Explanation: Accumulate sum while reading elements, then calculate average.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i;
  float sum = 0, average;

  printf("Enter array size (max 100): ");
  scanf("%d", &N);

  int T[100];  // Assuming max 100 elements

  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);
    sum += T[i];
  }

  average = sum / N;

  printf("Sum = %.2f\n", sum);
  printf("Average = %.2f\n", average);

  return 0;
}
```

Solution

12. Find Min and Max in Array

[Back]

Algorithm

Steps:

1. Start
2. Read array `T[N]`
3. Initialize `min = T[0]`, `max = T[0]`
4. For `i = 1` to `N-1`:
   - If `T[i] < min` then `min = T[i]`
   - If `T[i] > max` then `max = T[i]`
5. Print `min` and `max`
6. End

Explanation: Assume first element is both min and max, then compare with remaining elements.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, min, max;

  printf("Enter array size: ");
  scanf("%d", &N);

  int T[N];

  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);
  }

  // Initialize min and max with first element
  min = max = T[0];

  // Find min and max
  for(i = 1; i < N; i++) {
    if(T[i] < min) min = T[i];
    if(T[i] > max) max = T[i];
  }

  printf("Minimum = %d\n", min);
  printf("Maximum = %d\n", max);

  return 0;
}

```

Solution

13. Check if Array is Sorted

[Back]

Algorithm

Steps:

1. Start
2. Read array `T[N]`
3. Initialize `sorted = 1` (true)
4. For `i = 0` to `N-2`:
   - If `T[i] > T[i+1]` then:
     - `sorted = 0` (false)
     - Break loop
5. If `sorted == 1` print "Sorted"
6. Else print "Not Sorted"
7. End

Explanation: Compare each element with its next element. If any element is greater than the next, array is not sorted.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, sorted = 1;

  printf("Enter array size: ");
  scanf("%d", &N);

  int T[N];

  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);
  }

  // Check if array is sorted in ascending order
  for(i = 0; i < N - 1; i++) {
    if(T[i] > T[i+1]) {
      sorted = 0;  // Not sorted
      break;
    }
  }

  if(sorted)
  printf("Array is sorted in ascending order.\n");
  else
  printf("Array is NOT sorted in ascending order.\n");

  return 0;
}
```

Solution

14. Count Positive, Negative, Zero

[Back]

Algorithm

Steps:

1. Start
2. Read array `T[N]`
3. Initialize counters: `positive = 0`, `negative = 0`, `zero = 0`
4. For `i = 0` to `N-1`:
   - If `T[i] > 0` then `positive++`
   - Else if `T[i] < 0` then `negative++`
   - Else `zero++`
5. Print counters
6. End

Explanation: Use if-else chain to categorize each element.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, positive = 0, negative = 0, zero = 0;

  printf("Enter array size: ");
  scanf("%d", &N);

  int T[N];

  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);

    if(T[i] > 0)
    positive++;
    else if(T[i] < 0)
    negative++;
    else
    zero++;
  }

  printf("\nStatistics:\n");
  printf("Positive numbers: %d\n", positive);
  printf("Negative numbers: %d\n", negative);
  printf("Zero values: %d\n", zero);

  return 0;
}

```

Solution

15. Sequential Search in Array

[Back]

Algorithm

Steps:

1. Start
2. Read array `T[N]`
3. Read search value `x`
4. Initialize `found = 0`, `position = -1`
5. For `i = 0` to `N-1`:
   - If `T[i] == x` then:
     - `found = 1`
     - `position = i`
     - Break loop
6. If `found == 1` print position
7. Else print "Not found"
8. End

Explanation: Linear search - check each element sequentially until value is found or end of array.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, x, found = 0, position = -1;

  printf("Enter array size: ");
  scanf("%d", &N);

  int T[N];

  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);
  }

  printf("Enter value to search: ");
  scanf("%d", &x);

  // Sequential search
  for(i = 0; i < N; i++) {
    if(T[i] == x) {
      found = 1;
      position = i;
      break;
    }
  }

  if(found)
  printf("Value %d found at position %d\n", x, position);
  else
  printf("Value %d not found in array\n", x);

  return 0;
}
```

Solution

16. Copy Array to Another Array

[Back]

Algorithm

Steps:

1. Start
2. Read array `A[N]`
3. Create array `B[N]`
4. For `i = 0` to `N-1`:
   - `B[i] = A[i]`
5. Display both arrays
6. End

Explanation: Simple element-by-element copy using a loop.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i;

  printf("Enter array size: ");
  scanf("%d", &N);

  int A[N], B[N];

  printf("Enter %d integers for array A:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &A[i]);
  }

  // Copy A to B
  for(i = 0; i < N; i++) {
    B[i] = A[i];
  }

  // Display arrays
  printf("\nArray A: ");
  for(i = 0; i < N; i++) printf("%d ", A[i]);

  printf("\nArray B (copy of A): ");
  for(i = 0; i < N; i++) printf("%d ", B[i]);

  printf("\n");
  return 0;
}
```

Solution

17. Reverse an Array

[Back]

Algorithm

Steps:

1. Start
2. Read array `T[N]`
3. For `i = 0` to `N/2 - 1`:
   - Swap `T[i]` with `T[N-1-i]`
4. Display reversed array
5. End

Explanation: Swap first with last, second with second-last, etc. Stop at the middle.

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, temp;

  printf("Enter array size: ");
  scanf("%d", &N);

  int T[N];

  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);
  }

  // Display original array
  printf("Original array: ");
  for(i = 0; i < N; i++) printf("%d ", T[i]);
  printf("\n");

  // Reverse the array
  for(i = 0; i < N/2; i++) {
    // Swap T[i] with T[N-1-i]
    temp = T[i];
    T[i] = T[N-1-i];
    T[N-1-i] = temp;
  }

  // Display reversed array
  printf("Reversed array: ");
  for(i = 0; i < N; i++) printf("%d ", T[i]);
  printf("\n");

  return 0;
}
```

Solution

18. Merge Two Arrays [Back]

Algorithm

Steps:

1. Start
2. Read array `A[N]` and array `B[M]`
3. Create array `C[N+M]`
4. Copy all elements of A to C (positions 0 to N-1)
5. Copy all elements of B to C (positions N to N+M-1)
6. Display merged array C
7. End

Explanation: First copy all elements from first array, then all elements from second array.

C Code:

```c
#include <stdio.h>
int main() {
  int N, M, i, j;

  // First array
  printf("Enter size of array A: ");
  scanf("%d", &N);
  int A[N];
  printf("Enter %d integers for array A:\n", N);
  for(i = 0; i < N; i++) scanf("%d", &A[i]);

  // Second array
  printf("Enter size of array B: ");
  scanf("%d", &M);
  int B[M];
  printf("Enter %d integers for array B:\n", M);
  for(i = 0; i < M; i++) scanf("%d", &B[i]);

  // Merge arrays
  int C[N + M];

  // Copy A to C
  for(i = 0; i < N; i++) {
    C[i] = A[i];
  }

  // Copy B to C
  for(j = 0; j < M; j++) {
    C[N + j] = B[j];
  }

  // Display merged array
  printf("Merged array C (size %d):\n", N + M);
  for(i = 0; i < N + M; i++) {
    printf("%d ", C[i]);
  }
  printf("\n");

  return 0;
}
```

4.3 III. Character Strings (19-24)

Solution

19. String Length (Manual) [Back]

Algorithm

Steps:

1. Start
2. Read string `S`
3. Initialize `length = 0`
4. While `S[length] != '\0'`:
   - `length++`
5. Print `length`
6. End

C Code:

```c
#include <stdio.h>
int main() {
  char S[100];
  int length = 0;

  printf("Enter a string: ");
  fgets(S, sizeof(S), stdin);

  while(S[length] != '\0') {
    if(S[length] == '\n') {
      S[length] = '\0';
      break;
    }
    length++;
  }

  length = 0;
  while(S[length] != '\0') {
    length++;
  }

  printf("Length of string = %d\n", length);
  return 0;
}
```

Solution

20. Count Vowels in String

[Back]

Algorithm

Steps:

1. Start
2. Read string `S`
3. Initialize `vowelCount = 0`
4. For each character `ch` in `S`:
   - Convert `ch` to lowercase
   - If `ch` is 'a', 'e', 'i', 'o', or 'u':
     - `vowelCount++`
5. Print `vowelCount`
6. End

C Code:

```c
#include <stdio.h>
int main() {
  char S[100];
  int i, vowelCount = 0;

  printf("Enter a string: ");
  fgets(S, sizeof(S), stdin);

  for(i = 0; S[i] != '\0'; i++) {
    char ch = S[i];
    if(ch >= 'A' && ch <= 'Z') {
      ch = ch + 32;
    }

    if(ch == 'a' || ch == 'e' || ch == 'i' ||
    ch == 'o' || ch == 'u') {
      vowelCount++;
    }
  }

  printf("Number of vowels = %d\n", vowelCount);
  return 0;
}

```

Solution

21. Palindrome Check

[Back]

Algorithm

Steps:

1. Start
2. Read string `S`
3. Find length `n`
4. Initialize `isPalindrome = 1`
5. For `i = 0` to `n/2 - 1`:
   - If `S[i] != S[n-1-i]` then:
     - `isPalindrome = 0`
     - Break
6. If `isPalindrome` print "Palindrome"
7. Else print "Not Palindrome"
8. End

C Code:

```c
#include <stdio.h>
int main() {
  char S[100];
  int i, n, isPalindrome = 1;

  printf("Enter a string: ");
  fgets(S, sizeof(S), stdin);

  n = 0;
  while(S[n] != '\0') {
    if(S[n] == '\n') {
      S[n] = '\0';
      break;
    }
    n++;
  }

  for(i = 0; i < n/2; i++) {
    if(S[i] != S[n-1-i]) {
      isPalindrome = 0;
      break;
    }
  }

  if(isPalindrome)
  printf("Palindrome.\n");
  else
  printf("Not a palindrome.\n");

  return 0;
}
```

Solution

22. Copy String (Manual)

[Back]

Algorithm

Steps:

1. Start
2. Read string `source`
3. Initialize `i = 0`
4. While `source[i] != '\0'`:
   - `dest[i] = source[i]`
   - `i++`
5. `dest[i] = '\0'`
6. Display both strings
7. End

C Code:

```c
#include <stdio.h>
int main() {
  char source[100], dest[100];
  int i = 0;

  printf("Enter a string: ");
  fgets(source, sizeof(source), stdin);

  while(source[i] != '\0') {
    if(source[i] == '\n') {
      source[i] = '\0';
      break;
    }
    i++;
  }

  i = 0;
  while(source[i] != '\0') {
    dest[i] = source[i];
    i++;
  }
  dest[i] = '\0';

  printf("Source: %s\n", source);
  printf("Copy: %s\n", dest);

  return 0;
}
```

Solution

23. Compare Strings (Manual)

[Back]

Algorithm

Steps:

1. Start
2. Read strings `S1` and `S2`
3. Initialize `i = 0`
4. While `S1[i] != '\0'` and `S2[i] != '\0'`:
   - If `S1[i] != S2[i]` then:
     - Strings are different
     - Break
   - `i++`
5. If both strings ended at same time and all characters matched:
   - Strings are equal
6. Else:
   - Strings are different
7. End

C Code:

```c
#include <stdio.h>
int main() {
  char S1[100], S2[100];
  int i = 0, equal = 1;

  printf("Enter first string: ");
  fgets(S1, sizeof(S1), stdin);
  printf("Enter second string: ");
  fgets(S2, sizeof(S2), stdin);

  while(S1[i] != '\0' && S2[i] != '\0') {
    if(S1[i] == '\n') S1[i] = '\0';
    if(S2[i] == '\n') S2[i] = '\0';

    if(S1[i] != S2[i]) {
      equal = 0;
      break;
    }
    i++;
  }

  if(equal && S1[i] == S2[i])
  printf("Strings are equal.\n");
  else
  printf("Strings are NOT equal.\n");

  return 0;
}

```

Solution

24. Convert String to Uppercase

[Back]

Algorithm

Steps:

1. Start
2. Read string `S`
3. For each character `ch` in `S`:
   - If `ch` is lowercase ('a' to 'z'):
     - Convert to uppercase: `ch = ch - 32`
4. Display converted string
5. End

C Code:

```c
#include <stdio.h>
int main() {
  char S[100];
  int i;

  printf("Enter a string: ");
  fgets(S, sizeof(S), stdin);

  for(i = 0; S[i] != '\0'; i++) {
    if(S[i] == '\n') {
      S[i] = '\0';
      break;
    }
  }

  printf("Original: %s\n", S);

  for(i = 0; S[i] != '\0'; i++) {
    if(S[i] >= 'a' && S[i] <= 'z') {
      S[i] = S[i] - 32;
    }
  }

  printf("Uppercase: %s\n", S);
  return 0;
}

```

4.4 IV. Structures / Records (25-30)

Solution

25. Student Structure Definition

[Back]

Algorithm

Steps:

1. Define structure `Student` with:
   - `name[50]`
   - `firstName[50]`
   - `average` (float)
2. Declare variable `s` of type `Student`
3. Read data into `s`
4. Display `s` information
5. End

C Code:

```c
#include <stdio.h>
struct Student {
  char name[50];
  char firstName[50];
  float average;
};

int main() {
  struct Student s;

  printf("Name: ");
  fgets(s.name, sizeof(s.name), stdin);
  printf("First Name: ");
  fgets(s.firstName, sizeof(s.firstName), stdin);
  printf("Average: ");
  scanf("%f", &s.average);

  printf("\nStudent Information:\n");
  printf("Name: %s", s.name);
  printf("First Name: %s", s.firstName);
  printf("Average: %.2f\n", s.average);

  return 0;
}

```

Solution

26. Array of Students

[Back]

Algorithm

Steps:

1. Define `Student` structure
2. Read number of students `N`
3. Declare array `class[N]` of type `Student`
4. For `i = 0` to `N-1`:
   - Read data for `class[i]`
5. Display all students
6. End

C Code:

```c
#include <stdio.h>
struct Student {
  char name[50];
  float average;
};

int main() {
  struct Student class[100];
  int N, i;

  printf("Enter number of students: ");
  scanf("%d", &N);
  getchar();

  for(i = 0; i < N; i++) {
    printf("\nStudent %d:\n", i+1);
    printf("Name: ");
    fgets(class[i].name, sizeof(class[i].name), stdin);
    printf("Average: ");
    scanf("%f", &class[i].average);
    getchar();
  }

  printf("\nList of Students:\n");
  for(i = 0; i < N; i++) {
    printf("%d. %s - %.2f\n", i+1, class[i].name, class[i].average);
  }

  return 0;
}
```

Solution

27. Search Student by Name

[Back]

Algorithm

Steps:

1. Define `Student` structure and array
2. Read search name `targetName`
3. For each student in array:
   - If `student.name == targetName`:
     - Display student info
     - Set `found = true`
4. If not found, display message
5. End

C Code:

```c
#include <stdio.h>
#include <string.h>
struct Student {
  char name[50];
  float average;
};

int main() {
  struct Student class[4] = {
    {"Ali", 15.5},
    {"Mohamed", 12.0},
    {"Fatima", 18.0},
    {"Ahmed", 14.5}
  };
  char targetName[50];
  int i, found = 0;

  printf("Enter student name to search: ");
  fgets(targetName, sizeof(targetName), stdin);

  for(i = 0; i < 4; i++) {
    if(strcmp(class[i].name, targetName) == 0) {
      printf("Student Found:\n");
      printf("Name: %s", class[i].name);
      printf("Average: %.2f\n", class[i].average);
      found = 1;
      break;
    }
  }

  if(!found) {
    printf("Student not found.\n");
  }

  return 0;
}
```

Solution

28. Find Student with Highest Average

[Back]

Algorithm

Steps:

1. Define `Student` array
2. Initialize `maxIndex = 0`
3. For `i = 1` to `N-1`:
   - If `class[i].average > class[maxIndex].average`:
     - `maxIndex = i`
4. Display `class[maxIndex]`
5. End

C Code:

```c
#include <stdio.h>
struct Student {
  char name[50];
  float average;
};

int main() {
  struct Student class[100];
  int N, i, bestIndex = 0;

  printf("Enter number of students: ");
  scanf("%d", &N);
  getchar();

  for(i = 0; i < N; i++) {
    printf("\nStudent %d:\n", i+1);
    printf("Name: ");
    fgets(class[i].name, sizeof(class[i].name), stdin);
    printf("Average: ");
    scanf("%f", &class[i].average);
    getchar();

    if(class[i].average > class[bestIndex].average) {
      bestIndex = i;
    }
  }

  printf("\nBest Student:\n");
  printf("Name: %s", class[bestIndex].name);
  printf("Average: %.2f\n", class[bestIndex].average);

  return 0;
}

```

Solution

29. Sort Students by Average

[Back]

Algorithm

Steps:

1. Define `Student` array
2. Use Bubble Sort algorithm:
   - For `i = 0` to `N-2`:
     - For `j = 0` to `N-i-2`:
       - If `class[j].average < class[j+1].average`:
         - Swap `class[j]` and `class[j+1]`
3. Display sorted array
4. End

C Code:

```c
#include <stdio.h>
struct Student {
  char name[50];
  float average;
};
int main() {
  struct Student class[100], temp;
  int N, i, j;
  printf("Enter number of students: ");
  scanf("%d", &N);
  getchar();
  for(i = 0; i < N; i++) {
    printf("\nStudent %d:\n", i+1);
    printf("Name: ");
    fgets(class[i].name, sizeof(class[i].name), stdin);
    printf("Average: ");
    scanf("%f", &class[i].average);
    getchar();
  }
  for(i = 0; i < N-1; i++) {
    for(j = 0; j < N-i-1; j++) {
      if(class[j].average < class[j+1].average) {
        temp = class[j];
        class[j] = class[j+1];
        class[j+1] = temp;
      }
    }
  }
  printf("\nStudents Sorted by Average:\n");
  for(i = 0; i < N; i++) {
    printf("%d. %s - %.2f\n", i+1, class[i].name, class[i].average);
  }
  return 0;
}

```

Solution

30. Phonebook Directory

[Back]

Algorithm

Steps:

1. Define `Contact` structure with:
   - `name[50]`
   - `phone[20]`
2. Declare array `directory[MAX]`
3. Create menu:
   - 1. Add contact
   - 2. Display all contacts
   - 0. Exit
4. Implement each operation
5. End

C Code:

```c
#include <stdio.h>
struct Contact {
  char name[50];
  char phone[20];};
int main() {
  struct Contact directory[100];
  int choice, count = 0, i;
  do {printf("\nPhonebook Directory\n");
    printf("1. Add Contact\n");
    printf("2. Display All Contacts\n");
    printf("0. Exit\n");
    printf("Your choice: ");
    scanf("%d", &choice);
    getchar();
    switch(choice) {
      case 1:
      printf("\nAdd New Contact\n");
      printf("Name: ");
      fgets(directory[count].name, sizeof(directory[count].name), stdin);
      printf("Phone: ");
      fgets(directory[count].phone, sizeof(directory[count].phone), stdin);
      count++;
      printf("Contact added.\n");
      break;
      case 2:
      printf("\nAll Contacts:\n");
      for(i = 0; i < count; i++) {
        printf("%d. %s - %s", i+1, directory[i].name, directory[i].phone);}
      break;
      case 0:
      printf("Goodbye!\n");
      break;
      default:
      printf("Invalid choice.\n");
    }
  } while(choice != 0);
  return 0;
  }

```

5 The Exam Solutions

Solution

EXERCISE 1: Array Order Analysis [Back]

Problem Statement: Determine if an array is strictly increasing, strictly decreasing, or not ordered.

Algorithm

Steps:

1. Start
2. Read array size `N` and array `T[N]`
3. Initialize flags: `isInc = 1`, `isDec = 1`
4. For `i = 0` to `N-2`:
   - If `T[i] >= T[i+1]` then `isInc = 0`
   - If `T[i] <= T[i+1]` then `isDec = 0`
5. Check flags:
   - If `isInc == 1`: "Strictly Increasing"
   - Else if `isDec == 1`: "Strictly Decreasing"
   - Else: "Not Ordered"
6. End

C Code:

```c
#include <stdio.h>
int main() {
  int N, i;
  printf("Enter array size N: ");
  scanf("%d", &N);
  int T[N];
  printf("Enter %d integers:\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &T[i]);
  }
  int isIncreasing = 1;
  int isDecreasing = 1;
  for(i = 0; i < N - 1; i++) {
    if(T[i] >= T[i+1]) {
      isIncreasing = 0;
    }
    if(T[i] <= T[i+1]) {
      isDecreasing = 0;
    }
  }
  printf("\nResult: ");
  if(isIncreasing) {
    printf("STRICTLY INCREASING\n");
  } else if(isDecreasing) {
    printf("STRICTLY DECREASING\n");
  } else {
    printf("NOT ORDERED\n");
  }
  return 0;
}

```

Solution

EXERCISE 2: Longest Word in Sentence [Back]

Problem Statement: Find and display the longest word in a sentence.

Algorithm

Steps:

1. Start
2. Read sentence `s`
3. Initialize:
   - `maxLen = 0`
   - `maxStart = 0`
   - `currLen = 0`
   - `currStart = 0`
4. For each character `i` from 0 to end of string:
   - If `s[i]` is not space and not null:
     - If `currLen == 0`, set `currStart = i`
     - `currLen++`
   - Else (space or end of string):
     - If `currLen > maxLen`:
       - `maxLen = currLen`
       - `maxStart = currStart`
     - Reset `currLen = 0`
5. Extract and display word from `maxStart` with length `maxLen`
6. End

C Code:

```c
#include <stdio.h>
int main() {
  char sentence[200];
  int i, maxLen = 0, maxStart = 0;
  int currLen = 0, currStart = 0;

  printf("Enter a sentence: ");
  fgets(sentence, sizeof(sentence), stdin);

  for(i = 0; ; i++) {
    if(sentence[i] != ' ' && sentence[i] != '\0' && sentence[i] != '\n') {
      if(currLen == 0) {
        currStart = i;
      }
      currLen++;
    } else {
      if(currLen > maxLen) {
        maxLen = currLen;
        maxStart = currStart;
      }
      currLen = 0;

      if(sentence[i] == '\0' || sentence[i] == '\n')
      break;
    }
  }

  printf("Longest word: ");
  for(i = maxStart; i < maxStart + maxLen; i++) {
    printf("%c", sentence[i]);
  }
  printf("\nLength: %d\n", maxLen);

  return 0;
}

```

Solution

EXERCISE 3: Frequency Table [Back]

Problem Statement: Build frequency table and display values appearing more than once.

Algorithm

Steps:

1. Start
2. Read array size `N`
3. Initialize frequency array `freq[101]` all zeros
4. For `i = 0` to `N-1`:
   - Read value `val`
   - If `0 <= val <= 100`:
     - `freq[val]++`
5. Display values with frequency > 1
6. End

C Code:

```c
#include <stdio.h>
int main() {
  int N, i, value;
  int freq[101] = {0};

  printf("Enter number of elements: ");
  scanf("%d", &N);

  printf("Enter %d integers (0 to 100):\n", N);
  for(i = 0; i < N; i++) {
    scanf("%d", &value);

    if(value >= 0 && value <= 100) {
      freq[value]++;
    }
  }

  printf("\nValues appearing more than once:\n");
  int found = 0;
  for(i = 0; i <= 100; i++) {
    if(freq[i] > 1) {
      printf("Value %d appears %d times\n", i, freq[i]);
      found = 1;
    }
  }

  if(!found) {
    printf("No values appear more than once.\n");
  }

  return 0;
}
```

Solution

EXERCISE 4: Student Records Management [Back]

Problem Statement: Manage student records with various operations.

Algorithm

Steps:

1. Define `Student` structure
2. Read number of students `N`
3. For each student:
   - Read ID, Name, Average
   - Accumulate sum for class average
   - Track best student index
4. Display students with average 10
5. Calculate and display class average
6. Display best student information
7. End

C Code:

```c
#include <stdio.h>
struct Student {
  int id;
  char name[50];
  float average;
};

int main() {
  struct Student students[100];
  int N, i, bestIndex = 0;
  float classSum = 0, classAverage;

  printf("Enter number of students: ");
  scanf("%d", &N);
  getchar();

  for(i = 0; i < N; i++) {
    printf("\nStudent %d:\n", i + 1);
    printf("ID: ");
    scanf("%d", &students[i].id);
    getchar();
    printf("Name: ");
    fgets(students[i].name, sizeof(students[i].name), stdin);
    printf("Average: ");
    scanf("%f", &students[i].average);
    getchar();

    classSum += students[i].average;
    if(students[i].average > students[bestIndex].average) {
      bestIndex = i;
    }
  }

  printf("\nStudents with Average >= 10:\n");
  int countPassed = 0;
  for(i = 0; i < N; i++) {
    if(students[i].average >= 10.0) {
      printf("%d. %s (ID: %d) - %.2f\n",
      ++countPassed, students[i].name,
      students[i].id, students[i].average);
    }
  }

  classAverage = classSum / N;
  printf("\nClass Average: %.2f\n", classAverage);

  printf("\nBest Student:\n");
  printf("Name: %s", students[bestIndex].name);
  printf("ID: %d\n", students[bestIndex].id);
  printf("Average: %.2f\n", students[bestIndex].average);

  return 0;
}
```

Solution

EXERCISE 5: C Identifier Validation [Back]

Problem Statement: Check if a string is a valid C identifier.

Algorithm

Steps:

1. Start
2. Read string `identifier`
3. Check Rule 1: First character must be letter or underscore
4. For remaining characters:
   - Check Rule 2: Must be letter, digit, or underscore
5. If all rules satisfied: Valid
6. Else: Invalid
7. End

C Code:

```c
#include <stdio.h>
int main() {
  char identifier[50];
  int i, valid = 1;

  printf("Enter a string: ");
  fgets(identifier, sizeof(identifier), stdin);

  if((identifier[0] >= 'a' && identifier[0] <= 'z') ||
  (identifier[0] >= 'A' && identifier[0] <= 'Z') ||
  identifier[0] == '_') {
    valid = 1;
  } else {
    valid = 0;
  }

  i = 1;
  while(identifier[i] != '\0' && identifier[i] != '\n') {
    if(!((identifier[i] >= 'a' && identifier[i] <= 'z') ||
    (identifier[i] >= 'A' && identifier[i] <= 'Z') ||
    (identifier[i] >= '0' && identifier[i] <= '9') ||
    identifier[i] == '_')) {
      valid = 0;
      break;
    }
    i++;
  }

  if(valid) {
    printf("VALID C Identifier\n");
  } else {
    printf("INVALID C Identifier\n");
  }

  return 0;
}
```

Appendix: Document Summary and References

Summary

This document serves as a comprehensive instructional guide and exercise compendium for the foundational topics in computer programming and algorithm design, with a focus on the C programming language. Structured into four core chapters, it systematically covers essential programming constructs, data structures, and problem-solving techniques. The material progresses from basic instructions and control flow, through arrays and string manipulation, to the organization of data using structures. Each section is reinforced by a curated set of practice exercises, complete with detailed algorithmic descriptions and implemented C code solutions. The included examination problems provide a synthesis of the covered concepts, testing the learner's ability to apply knowledge to more complex, integrated tasks. The guide adopts a clear, pedagogical approach, combining theoretical explanation with practical implementation, thereby supporting both conceptual understanding and skill development.

Last Word

This guide reaches its conclusion after presenting a structured and comprehensive learning pathway that combines methodological explanation with practical implementation of programming and algorithmic concepts. It was carefully prepared to serve as a reliable academic resource, supporting learners throughout their educational journey and helping them establish a solid foundation in algorithmic thinking and software development.
If this work has contributed in any way to clarifying concepts or enhancing the learning experience, then its intended purpose has been fulfilled. It is sincerely hoped that the knowledge gained from this guide will be beneficial in future academic and professional endeavors. With best wishes for continued success in learning, and with respectful appreciation extended to all those who supported and inspired this work.

References

References

[1] Ben Janna, H. (2025).*Lecture Notes and Exercises in Algorithms and Data Structures 01.* Echahid Cheikh Larbi Tebessi.

[2] Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language* (2nd ed.). Prentice Hall.

[3] Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.

[4] Knuth, D. E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3rd ed.). Addison-Wesley.

[5] Skiena, S. S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer.

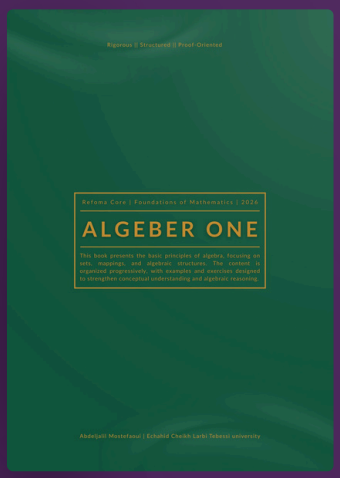
[6] Deitel, P. J., & Deitel, H. M. (2016). *C How to Program* (8th ed.). Pearson.

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

To request a design, contact us via Telegram: **@jaliiilo**

Other books by us :

Portfolio


Rigorous || Structured || Proof-Oriented
Reforma Core | Foundations of Mathematics | 2026
ALGEBER ONE
This book presents the basic principles of algebra, focusing on sets, mappings, and algebraic structures. The content is organized progressively, with examples and exercises designed to strengthen conceptual understanding and algebraic reasoning.
Abdeljalil Mostefaoui | Echahid Cheikh Larbi Tebessi university

Rigorous || Structured || Proof-Oriented
Reforma Core | Foundations of Mathematics | 2026
ANALYSIS ONE
This book introduces the fundamental concepts of mathematical analysis, including limits, continuity, and differentiation. It adopts a rigorous and structured approach, supported by examples and detailed exercises, to help students develop logical reasoning and a solid analytical foundation.
Abdeljalil Mostefaoui | Echahid Cheikh Larbi Tebessi university